



UNIVERSIDADE FEDERAL DE ALAGOAS
CAMPUS A.C SIMÕES
CIÊNCIA DA COMPUTAÇÃO

JONATAN LEITE ALVES

Relatório do Projeto Compiladores

Interpretador MiniPar

Maceió - AL

Objetivo:

Implementar um interpretador aplicando os conceitos da disciplina de Compiladores.

1. Enunciado

Implementação de um interpretador MiniPar, utilizando os conceitos estudados na disciplina de Compiladores. O interpretador implementado deve ser capaz de analisar e executar programas escritos na linguagem MiniPar, ao qual inclui estruturas sequenciais, paralelas (threads), condicionais e de repetição, além de operações aritméticas básicas e comunicação entre processos através de um canal de comunicação (cliente-servidor), implementado com sockets.

2. Introdução

O desenvolvimento de compiladores e interpretadores é uma das partes fundamentais para se obter entendimento de como as linguagens de programação funcionam. Além disso, proporciona uma melhor compreensão aos desenvolvedores de como as linguagens são traduzidas e executadas pelos computadores. Neste projeto acadêmico da disciplina de Compiladores, aplicamos os conhecimentos adquiridos na implementação de um interpretador para a linguagem MiniPar.

A linguagem MiniPar foi projetada para demonstrar os principais conceitos de programação concorrente, paralela e distribuída, sendo de certa forma, uma linguagem simples, mas que permite a execução de programas com diferentes fluxos de execução, além de permitir comunicação entre sistemas distribuídos, localmente ou na rede (estrutura cliente-servidor). O interpretador deverá ser capaz de analisar programas escritos nessa linguagem e executá-los de acordo com as especificações da linguagem.

No decorrer deste relatório, apresentaremos a gramática da linguagem MiniPar, a arquitetura de software do interpretador, os pseudocódigos dos analisadores léxico e sintático, por fim alguns tratamentos de erros implementados, detalhes sobre a linguagem de programação e ambiente de desenvolvimento utilizados, e por fim, algumas demonstrações da execução do interpretador com programas de teste. Para ter uma melhor noção de funcionamento, é aconselhável fazer uma experimentação da linguagem. Para isto, ao fim do relatório terá o link do repositório github do projeto.

3. Gramática

Produções

programa_minipar : bloco_stmt

bloco_stmt : bloco_SEQ
 | bloco_PAR
 | bloco_stmt bloco_SEQ
 | bloco_stmt bloco_PAR

bloco_SEQ : SEQ stmts

bloco_PAR : PAR stmts

stmts : stmt
 | stmts stmt

stmt : atribuicao
 | bloco_IF
 | bloco_WHILE
 | bloco_INPUT
 | bloco_OUTPUT
 | c_channel
 | c_channel_stmt

atribuicao : ID = expr
 | ID = STRING
 | ID = bloco_INPUT

expr : INT
 | ID
 | STRING
 | expr + expr
 | expr - expr
 | expr * expr
 | expr / expr
 | expr < expr
 | expr > expr
 | expr <= expr
 | expr >= expr
 | expr == expr
 | expr != expr

bool : expr

```

c_channel : C_CHANNEL ID ( STRING , STRING )

c_channel_stmt : send_stmt
                | receive_stmt
                | receive_result_stmt

send_stmt : ID . send ( ID , expr , expr , expr )
           | ID . send ( ID )

receive_stmt : ID . receive ( ID , expr , expr , expr )
              | ID . receive ( ID )

bloco_IF : IF ( bool ) { stmts }

bloco_WHILE : While ( bool ) { stmts }

bloco_INPUT : Input ( )

bloco_OUTPUT : Output ( output_args )

output_args : expr
             | output_args , expr

```

4. Arquitetura

O interpretador MiniPar foi dividido em 4 módulos:

- **Módulo de Entrada (main):** Responsável por ler o código fonte do programa MiniPar. Também é responsável por determinar o fluxo da análise e execução do programa.
- **Analizador Léxico:** Realiza a análise léxica do código fonte, identificando os tokens e seus atributos.
- **Analizador Sintático:** Realiza a análise sintática do código fonte, verificando se ele está de acordo com a gramática da linguagem MiniPar.
- **Executor:** executa as instruções do programa MiniPar, seguindo as regras da linguagem MiniPar

5. Pseudocódigo do Analisador Léxico

Importação dos módulos necessários

Inicializa a lista de tokens

Inicializa as expressões regulares para os tokens

#trata tokens especiais a seguir

Função t_ID:

Identifica tokens do tipo ID

Verifica se o token está entre os tokens reservados

Retorna o token identificado

Função t_INT:

Identifica tokens do tipo INT

Retorna o token identificado

Função t_STRING:

Identifica tokens do tipo STRING

Retorna o token identificado

Função t_ignore:

Ignora espaços em branco, tabulações e quebras de linha

Função t_error:

Identifica caracteres ilegais

Exibe uma mensagem de erro com a linha e o caractere ilegal

Muda o bool da variavel de controle erros para True (indicando algum erro encontrado)

Continua a análise ignorando o caractere ilegal

Função t_COMMENT:

Identifica comentários

Ignora os comentários encontrados

Cria analisador léxico pronto para uso

6. Pseudocódigo do Analisador Sintático

Inicializa as precedências dos operadores

Função p_programa_minipar:

- Define a produção para o programa miniPar
- Recebe como entrada um bloco de declarações
- Retorna o bloco de declarações

Função p_bloco_stmt:

- Define a produção para um bloco de instruções
- Pode ser um bloco SEQ, um bloco PAR ou uma combinação de blocos
- Retorna o bloco de instruções

Função p_bloco_SEQ:

- Define a produção para um bloco SEQ
- Recebe uma sequência de instruções
- Retorna o bloco SEQ com as instruções

Função p_bloco_PAR:

- Define a produção para um bloco PAR
- Recebe um paralelismo de declarações
- Retorna o bloco PAR com as declarações

Função p_stmts:

- Define a produção para uma sequência de instruções
- Pode ser uma única instrução ou várias instruções em sequência
- Retorna a lista de instruções

Função p_stmt:

- Define a produção para uma instrução
- Pode ser uma atribuição, um bloco IF, um bloco WHILE, etc.
- Retorna a declaração

Função p_atribuicao:

- Define a produção para uma atribuição
- Recebe um identificador e o associa a uma expressão ou string podendo vir de um input
- Retorna a atribuição

Função p_expr:

- Define a produção para uma expressão
- Pode ser um número, uma variável ou uma operação matemática
- Retorna a expressão

Função p_bool:

Define a produção para uma expressão booleana

Recebe uma expressão

Retorna a expressão booleana

Função p_c_channel:

Define a produção para um canal de comunicação

Recebe um identificador e dois componentes (endereço dos envolvidos na comunicação)

Retorna o canal de comunicação

Função p_c_channel_stmt:

Define a produção para uma instrução de canal de comunicação

Pode ser uma instrução de envio ou recebimento

Retorna a instrução do canal de comunicação

Função p_send_stmt:

Define a produção para uma instrução de envio

Possibilidade1: Recebe nome do canal (identificador), operação envio no canal, operação matemática, 2 números operandos, por fim variável que obtém resultado

Possibilidade 2: Recebe nome do canal, a operação de envio no canal e identificador do resultado

Retorna a instrução de envio

Função p_receive_stmt:

Define a produção para uma instrução de recebimento

possibilidade 1: Recebe nome do canal (identificador), operação de recebimento no canal, operação matemática, 2 números operando, por fim variável que obtém resultado

possibilidade 2: Recebe nome do canal, a operação de recebimento no canal e identificador do resultado

Retorna a instrução de recebimento

Função p_error:

Trata os erros sintáticos

Marca o flag de erro no Executor

Exibe uma mensagem de erro com a linha e o token

Recupera a análise e continua

Cria o analisador sintático, utilizando-se da análise léxica, para obtenção dos tokens

7. Linguagem de Programação e Ambiente de Desenvolvimento:

O interpretador MiniPar foi implementado na linguagem Python, utilizando-se da biblioteca PLY. A PLY tem por finalidade simplificar a análise léxica e sintática criando analisadores a partir de especificações em Python, facilitando todo o processo. Essa biblioteca se utiliza de expressões regulares e gramáticas livres de contexto para gerar os analisadores, logo permite que as regras sejam facilmente especificadas em Python.

Para a implementação completa do Interpretador MiniPar foi utilizado como ambiente de desenvolvimento o Visual Studio Code.

8. Tratamento de erros

O interpretador da linguagem MiniPar possui um sistema de tratamento de erros que abrange tanto erros léxicos quanto sintáticos. Esse tratamento é fundamental para garantir que o interpretador seja capaz de lidar adequadamente com entradas inválidas, fornecendo mensagens de erro significativas para o usuário.

Antes de tudo, é importante entender o papel da variável `has_error` pertencente ao módulo `Executor.py`. Ela é inicializada como `False`, indicando a ausência de erros. Contudo, ao detectar um erro léxico ou sintático, seu valor é alterado para `True`, indicando a presença de erros e impedindo a execução do programa. O interpretador MiniPar utiliza as funções `t_error()` (para erros léxicos) e `p_error()` (para erros sintáticos) para lidar com essas situações. Essas funções tentam se recuperar após um erro encontrado e em seguida continuam a análise do código em busca de mais erros por todo o programa. No entanto, para evitar que o programa seja executado com erros, uma verificação condicional da variável `has_error` atua como uma barreira.

8.1 Erros Léxicos

O tratamento de erros léxicos é realizado pelo analisador léxico, que identifica tokens inválidos ou malformados. Quando um erro léxico é detectado, o analisador interrompe a análise e emite uma mensagem de erro indicando a posição do erro no código fonte. Após identificar o erro, o analisador continua a análise a partir do próximo token válido.

Exemplo de Mensagem de Erro:

```
Caractere ilegal ';' na linha 1
```

8.2 Erros Sintáticos

Os erros sintáticos são tratados pelo analisador sintático, que identifica estruturas gramaticais incorretas no código fonte. Quando um erro sintático é detectado, o analisador interrompe a análise e emite uma mensagem de erro indicando a posição do erro e, quando

possível, uma sugestão de correção. Após identificar o erro, o analisador tenta se recuperar e continuar a análise a partir do próximo ponto de sincronização na gramática.

Exemplo de mensagem de Erro:

```
Erro sintático na linha 1, token '{'
```

8.3 Erros semânticos

O projeto inicialmente não previa lidar com erros semânticos, porém o erro semântico de utilizar uma variável sem declaração prévia foi tratado. Assim, sempre que for necessário utilizar uma variável, é imprescindível declará-la previamente. Veja abaixo 2 exemplos de tratamento de erros. No primeiro mostra o comportamento ao lidar com uma variável *val* que não foi declarada, e no segundo mostra que é errado usar um canal (*calc* e.g), sem antes o ter declarado.

Exemplo de mensagem de erro 1:

```
Erro semântico: identificador val não declarado
```

Exemplo de mensagem de erro 2:

```
Erro semântico: identificador 'calc' em 'calc.send()' não declarado
```

9. Implementação

O código-fonte, incluindo todos os módulos do interpretador, está disponível no [repositório do GitHub do projeto](#). Além disso, juntamente ao código-fonte, foram disponibilizados 3 programas escritos na linguagem MiniPar, que servirão como testes para as principais abordagens da linguagem: comunicação por sockets, paralelismo com threads, laços de repetição, condicionais, etc.

10. Execução programas de testes em MiniPar

Programa 1:

O programa 1 é um exemplo de comunicação entre cliente e servidor para uma calculadora aritmética simples. O [cliente](#) (computador 1) solicita uma operação aritmética e envia os operandos para o [servidor](#) (computador 2), que realiza o cálculo e retorna o resultado ao cliente. O programa utiliza um canal de comunicação chamado "calc" para troca de mensagens entre os computadores. O cliente exibe as opções de operações aritméticas, lê a operação desejada e os operandos, envia esses dados para o servidor, o servidor faz o cálculo e depois devolve o resultado para o cliente. Por fim, o cliente recebe o resultado e o exibe na tela.

```
jhonathanmilk@jotamilk: ~/interpretador-minipar
jhonathanmilk@jotamilk:~/interpretador-minipar$ minipar program1-server.mp
Aguardando conexões

Conexão estabelecida
Operação: 520*2

Resultado: 1040

Enviando resultado
jhonathanmilk@jotamilk:~/interpretador-minipar$
```

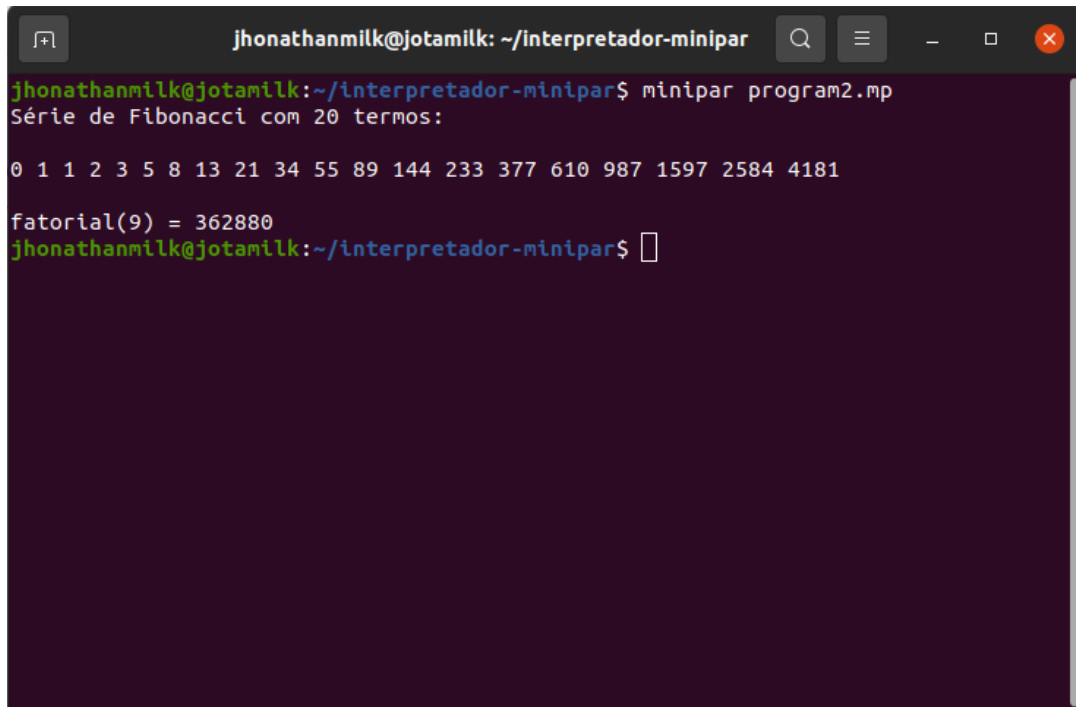
```
jhonathanmilk@jotamilk: ~/interpretador-minipar
jhonathanmilk@jotamilk:~/interpretador-minipar$ minipar program1-client.mp
CALCULADORA

Operadores: + - * /

Digite o primeiro numero:
520
Digite o segundo numero
2
Digite a operacao entre os números
*
resultado: 1040
jhonathanmilk@jotamilk:~/interpretador-minipar$
```

Programa 2:

O [programa 2](#) é um exemplo de implementação paralela em linguagem MiniPar para calcular o fatorial de um número e a série de Fibonacci de forma simultânea. Ele utiliza o conceito de execução paralela (implementado com o uso de threads em Python) para realizar os cálculos de forma concorrente no mesmo computador. A seção PAR indica que as instruções seguintes serão executadas de forma paralela. Então, ambas as threads são executadas de forma simultânea, calculando o fatorial de um número e a série de Fibonacci, respectivamente, no mesmo computador.



```
jhonathanmilk@jotamilk: ~/interpretador-minipar
jhonathanmilk@jotamilk:~/interpretador-minipar$ minipar program2.mp
Série de Fibonacci com 20 termos:

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

fatorial(9) = 362880
jhonathanmilk@jotamilk:~/interpretador-minipar$
```